Operations in a loop.

Single-pass random optimal selection.

tsh73, may 2013

Table of Contents

(How come?)

I encountered this problem in one of my programs.
I thought solution to be kind of cool, in a nerdy kind of way; but I had a doubt if it’s too obvious to warrant an article of its own. So I checked it on my colleague, a teacher; and she said “I think it’s interesting, go ahead and post it”. So here I am.

The problem:

Imagine you have a choosing program, likely involving big (lengthy) search. You are interested in picking just one of “best” solutions; being “best” amounts to having maximum of some criteria – a single number. However, beforehand you have no idea what number would it be; more, you have no idea if it would be single solution with maximum criteria or hundreds of them.
But in case you have several of them, you think it would be nice to pick one of them at random – I mean, with equal probability. (In case you program some game, say Tic-tac-toe, it would make computer opponent play more interesting – instead of picking same response along each path, it would show some variation). (Note: You can easily have first of them or last of them, depending of putting > or >= in your maximum search code; but with equal probability?)

Easy (read no-thinking) way

So. Easy (read no-thinking) way if doing this would require several passes of search loop.

- 1. On the first pass, you determine maximum criteria value (“best”).
- 2. On the second pass, you count number of occurrences (Nbest) of that “best” value.

- Then you can dimension array for that value (dim A(Nbest))
  3. Third pass fill that array with solutions yielding “best” criteria.
- And at last, we can finally generate random number R between 1 and Nbest and return A(R).

```
'problem:
'You have a choosing program, likely involving big (lengthy) search.
'You are interested in picking just one of "best" solutions;
'being "best" amounts to having maximum of some criteria - a single nu
mber.
'v.1. Easy (read no-thinking) way

'for now, i 1..N would be search space, b(i) is criteria
'i, so that b(i)= max, is solution (one of).
N=100
dim b(N)
NN=int(rnd(0)*20)+20
for i = 1 to N
    b(i)=int(rnd(0)*NN)
next
print "We have ";N; " integer random numbers."

'On the first pass, you determine maximum criteria value ("best").
best = -1e40  'so that it would be guaranteed less then maximum
for i = 1 to N
    if b(i)>best then best=b(i)
next
print "Maximum of them: ";  best
'On the second pass, you count number of occurrences (Nbest)  of that
"best"  value.
Nbest=0
print "solution","value"
for i = 1 to N
    if b(i)=best then
        Nbest=Nbest+1
        print i, b(i)
    end if
next
print "---------------------"
print "Num of solutions(maxumums) ";Nbest
'Then you can dimension array for that value (dim A(Nbest))
dim a(Nbest)
'Third pass fill that array with solutions yielding "best" criteria.
j=0
print "number","solution","value"
for i = 1 to N
    if b(i)=best then
        j=j+1
        a(j)=i
        print j, a(j), b(a(j))
    end if
next
print "---------------------"
'And at last, we can finally generate random number R between 1 and Nb
est
R=int(rnd(0)*Nbest)+1
'and return A(R).
print "We choose solution ";a(R);" with value ";b(a(R))
```

This approach has one definite value – clarity. Remember?
Make it work – make it work right – make it work fast, and only in that order!
But supposed you already at "work right" phase, some speeding up might as well be welcome. We started with "big (lengthy) search", remember?

Cutting it 2x

Currently, our algorithm passes over that lengthy search trice.
We can rather easily cut it 2x:
First, we can combine first two phases in one, in a single pass

```
'On the first pass, you determine maximum criteria value ("best").
'On the second pass, you count number of occurrences (Nbest)  of that
"best"  value.
'   combine first two phases in one, in a single pass
best = -1e40  'so that it would be guaranteed less then maximum
Nbest=0
for i = 1 to N
```

```
    if b(i)=best then
        Nbest=Nbest+1
    end if
    if b(i)>best then
'order is important: it's too late to check if(b(i)=best)
        best=b(i)       'after assignment on this line!
        Nbest=1
    end if
next
print "Maximum of them: ";  best
print "Num of solutions(maxumums) ";Nbest
```

So that leaves us with two passes (of three)
Second, we can skip that array business altogether.
We just generate R then run last pass, stopping after hitting R-th best value.

```
'We just generate R then run last pass, stopping after hitting R-
th best value.
R=int(rnd(0)*Nbest)+1
print "We choose ";R;"-th solution"
j=0
print "number","solution","value"
for i = 1 to N
    if b(i)=best then
        j=j+1
        print j, i, b(i)
        if j=R then exit for    'stopping after hitting R-
th best value
    end if
next
print "----------------------"
print "We choose solution ";i;" with value ";b(i)
```

That will cost us from around 1 (best case, R=1) to around full search pass (worst case, R=Nbest), in average – a half. So we end up with 1½ of a full search. As I promised, 2x cut from three passes :).

And now, in a single pass

Could it be made better?
Well, yes.
We will do all that in a single pass.
Here’s how.

Then we hit best value, we could have a counter – how much times we already had that best value?

- If answer is 0, (this is best value and it happened just now), then we just keep it, no question asked.
- If answer is 1, we have N=2 equal values. So we take new value with probability 1/N
  (ha! It just means ½ in our case!), that is, with condition
    - `If rnd(0)`